Full Potential Local Orbital
Minimum Basis
Bandstructure Scheme

User's Manual

by

Klaus Koepernik

April 26, 2005

Contents

Abbreviations **V**

1 Introduction **1**
1.1 Program structure 1
1.2 Input structure 2

2 Version control **4**
2.1 Upgrading 4
2.2 Downgrading 5

3 Files **7**
3.1 General 7
3.2 File class 1 8
3.3 File class 2 12
3.4 File class 3 16
3.5 Directories 16

4 Automation, scripting, pipe-mode **17**
4.1 Rules 17
4.2 Syntax 19
4.3 How to set up automation 22
4.4 Advanced features 25
4.4.1 Basis equivalence classes 25
4.4.2 Initial polarization, inital spin split 26
4.5 Example 28

Appendix **37**
A FAQ 38
B Summary of Changes 41

Abbreviations

DFT:	density functional theory
L(S)DA:	local (spin) density approximation
L(S)DA+U:	local (spin) density approximation + U-functional
CPA:	coherent potential approximaiton
VCA:	virtual crystal approximaiton
LO:	local orbital
DOS:	density of states
FSM:	fixed spin moment (method)
`stdin`:	the unix standard input file/channel
`stdout`:	the unix standard output file/channel
`stderr`:	the unix standard error file/channel
`PID`:	the unix process ID
v^{ew}:	Ewald potential
v^{C}:	Coulomb potential
v^{H}:	Hartree potential
v^{at}:	Atom potential
v^{cf}:	Confining potential
v^{cryst}:	Effective crystall potential
KS:	Kohn-Sham
qn:	quantum number
xc:	exchange and correlation

Chapter 1

Introduction

This document shall help to understand the FPLO package. We adopt the following notions. Text in typewriter style refers to unix commands, FEDIT options, file content and such things. Text in blue typewriter are names of files. The special symbols FPLO, FEDIT, ... are placeholders for the actual (version-related) full names of these programs.

Example: You have installed the binary `fplo5.00-18` and related binaries. Then the unix command (example)

```
FEDIT  -pipe < ./=.pipefile 2>./+log 1>/dev/null
```

means

```
fedit5.00-18 -pipe < ./=.pipefile 2>./+log 1>/dev/null
```

for your installation.

The installation process creates links with the generic names `fplo`, `fedit` and so on, which point to the installed version. Hence, if various versions are installed, the generic names refer to the executables, which where installed last!

Advice: It is better to call the fully qualified executable names like `fplo5.00-18` instead of the generic names. Anyway, every important output produced by the FPLO package contains the version number of the program, which produced the output. Just look at it, if in doubt.

1.1 Program structure

The FPLO package consists of a the source tree, which you hopefully compiled succesfully, and of the executables, which reside in `$HOME/FPLO/bin` in the standard installation. See installation instructions in `FPLO.../install/README...` for more information. There are 5 binaries and a number of scripts. The binaries are the following:

FPLO The bandstructure solver for the Kohn-Sham problem of bulk systems. It is a monolithic program, which performs the whole self consistent calculation. There are no such things as a bundle of standalone programms, which handle subtasks of the whole problem. The input of FPLO is handled via the input editor.

FEDIT The input editor. It handles the input editing for FPLO, DIRAC and BANDPLOT. There is basic help available via

```
FEDIT -h
```

Furthermore, there are help screens for every menu, which explain a number of aspects. **Please read them.** FEDIT has a pipe-mode, which is designed for automatic input management. It allows manipulation of input files by scripts, without messing up the input structure.

BANDPLOT The bandstructure plotting utility. It is normally used by FEDIT (and not called from the command line) to create band-structure and band-weight plots. Call

```
FEDIT -bandplot
```

DIRAC The standalone atom solver (spherical atoms). This solves the relativistic DFT problem for spherical atoms. To edit the input, call

```
FEDIT -atom
```

To learn more read the help screens in this mode, . **SIC is not yet working!**

FPIOTEST A utility for input manipulations. This is mainly used by the scripts of the distribution.

1.2 Input structure

The input files use a special syntax, which has some aspects in common with the `C`-programming language. It is, however, not one of the common data languages around, but a private one. The reason to introduce this feature was the general design of the input handling and the aspired independence on external libraries, which can not be assumed to be present on every machine. The package contains an input parser, which is accessible to the `C` and `F90` code. This reduces the need of `FORTRAN` IO management and thus increases the flexibility of version management and such things.

As a consequence, the user normally cannot and **should not** alter the content of the input files. **All input settings are meant to be changed with the help of** FEDIT **only.** In fact FEDIT is very easy to handle and there is no need for manual input file manipulations. (Some scripts also change the input, using FPIOTEST, which in turn uses the mentioned parser to achieve this goal.) For batch jobs/scripting/automation there is a special mode called pipe-mode in FEDIT.

The input, which is needed by the various executables is managed in a particular way. The executables create communication files (`+fedit`, `+fedithelp`), which tell FEDIT how to process input and how to manage menus. FEDIT uses some methods of back-communication[1] to the executables to have them responding in certain ways. This kind of communication is designed to avoid the user to edit input files directly (with a few exceptions).

If FPLO is called in an empty directory, it will imediatly terminate with a message telling that it created one of those communication files. On the next invokation it creates standard input files and again terminates. All this is not seen by the user, if he uses FEDIT.

Important: To archive a calculation it is sufficient to archive all files of class 1 with the prefix '`=.`' (see Chapter 3). If one uses such an archived calculation later to, say, create some additional output data, the first call to FPLO would stop as described above (unless FEDIT was used before). So, dont be afraid if the code exits, stating that it created the communication files. Just restart it and everthing will be fine.

[1] These are command line options (BANDPLOT), status variables in the input files (=.sym) and deletion of certain files.

Chapter 2

Version control

The package uses rather strict version control rules. A version number has the form "x.xx", where x is a placeholder for a digit. Furthermore there is a release number, which has the form "x". A full version-release number is the version number followed by a "-" followed by the release number like in 5.00-18, which means version 5.00 release 18.

Every major input file contains its own version number. Every executable has its own internal version number. To avoid problems, one usually cannot use files with programs of different version.

There is a simple rule: **The version number of the code is changed whenever that its input has changed.** If only the release number has changed the input is not altered. Normally, changes of the input consist of adding something. In rare cases, the structure changes. For this reason it is not recommended to manipulate version numbers of files by hand.

2.1 Upgrading

If the version number of FPLO has been increased it is rather simple to upgrade the old files. FPLO upgrades the files in the working directory automatically if they have older version numbers. Just call FEDIT or FPLO and have a look at the output. If FEDIT is used it will ask the user before performing the upgrade, while FPLO will do it without asking!

Attention: We strongly recomment to copy the whole directory (assumed you organize different calculations in different directories) before executing the upgrade. The reason is that the numerical results between different versions may slightly differ due to numerical changes/improvements[1].

[1] After all FPLO (like any other code) solves the problem approximately, although rather accurate.

2.2 Downgrading

Sometimes, it is nessecary to downgrad files to an older version number, mostly to undo erronous upgrading of files belonging to a series of calculations, which is intended be completed with the older FPLO version. However, sometimes one may whish to recalulate something with an older version for comparison (although this should be a very rare case for the normal user).

Downgrading means that some information gets lost. Thus, you should consider to **save a backup copy** of the files **before** you **downgrade**. Have in mind that some modes are not available in older versions of FPLO. Downgrading of calculations using such mode makes no sense. Examples are the grouping of orbitals, the relativistic mode and the LSDA+U.

Downgrading is a bit more complicated than upgrading. There is a `perl` script in the distribution, which handles the complicated restructuring.

1. Execute `fdowngrad.pl` at the command line and answer a few questions. As a result the input files are downgraded. The old files are copied to a backup directory called `fdowngrad_backup[n]`, where `[n]` is a number which is increased on every call to the script.

2. Next call the older version of FEDIT to regularize the files. **NEVER SKIP THIS POINT!**

Attention: Downgrading from version $\geq$ `4.00` to version $<$ `4.00` includes a subtle step, which will be explained here in detail. The density file information has been enriched in version `4.00`. In particular, the mesh has changed. If one would now use the file as it is with an older FPLO version, the density would be interpreted with the wrong mesh type[2]. To avoid this it is nessecary to map the new density onto the old mesh, which can only be done with an FPLO version $\geq$ `4.00`. The nessecary steps are

1. Call FEDIT of your current version ($\geq$ `4.00`) and set the option `'EXPORT_V3_DENSFILE'`

2. Run the corresponding FPLO. It will stop after the conversion.

3. Now proceed with `fdowngrad.pl` as described above.

Important: `fdowngrad.pl` tries to do these steps for you. This will only work if you have the proper FPLO version, best the one which has the version number of the files to be downgraded. If everything fails, you still have the backup. In the worst case you need to re-iterate the density[3] .

Pitfalls: Some information from files of newer version may be invalid in older version executables. This is seen if the older version FEDIT is used on the downgraded files (at least on exit, there will be a message.). Correct the input and rethink if you really did what you wanted to do.

[2] This basically results in the destruction of the (already converged) density.
[3] :-(

Example:

Downgrading a full relativistic input say version `4.00` to version `3.00` (where full relativistic mode is not available) will leave the relativistic mode-data untouched in `=.in`. But, this is invalid input in `fplo3.00-6`. Calling `fedit3.00-6` and executing `quit/save` will result in an error message about an invalid value of the variable `relativistic`. Go to the `relativisitc`-select box and select a proper option. Now, you can `quit/save` and you have valid input.

Chapter 3

Files

3.1 General

The files used by FPLO are classified into 3 major classes.

1. Files, which are contain input or both input and output data and which are nesseccary for a succesful restart of a previously converged calculation. These have the prefix `'=.'`. One should not delete these files nor use them for different calculations with different parameter settings. A safe rule is: each calculation is done in a seperate directory. One may copy all these files into a new directory, modify the input with FEDIT and start a new calculation. This is especially usefull in the case of slight changes of some parameters, in which case the density of the previous calculation is usually a better starting point than the atomic density (created in the very beginning of a calculation if `=.dens` is absent). If, however, the number of sites in the unit cell or the type of elements is changing, an initial density is needed (see FPLO output).

2. Files, which are mainly output files and which are not nesseccary for a succesful restart of a previously converged calculation. These files have the prefix '`+`'. In general one can delete these files (`rm ./+*`) after successful calculations.

   Caution: the bandstructure is written into `+band` or `+bweights`. These files are used by BANDPLOT to create a postscript picture of the bandstructure. So, if one indavertently deletes these files one needs a single step calculation, starting from a converged density, to recreate them.

   Important: Many (new) unix tools will interprete the '`+`' sign as an option flag. To use these tools with '`+`'-files, specify `./+file` instead `+file` on command line! [1]

[1] When FPLO was designed, this habit of unix (Linux) tools was not yet widespread, and hence not recognized by the FPLO authors.

3. Files, which are pure output. These files have no prefix.

   Example: `bravais.ps, primitive.ps`

In the following the most important files are described in more detail.

3.2 File class 1

The prefix '`=.`' indicates their primary use as input to FPLO. Nevertheless, they are partially output files. FPLO uses 9 such files[2]:

`=.sym`	definition of crystal symmetry
`=.in`	all data (including a copy of `=.sym`), which control the calculation, except the basis
`=.basis`	the basis definition, including the compression parameters (x_0)
`=.dens`	the charge density in terms of local angular momentum components
`=.densgrid`	a grid on which to output charge density or potential, output to `+densgrid`
`=.ldos`	defines, which atoms orbital (l- and m-) resolved densities of states are plotted. **is obsolete since version 4.01**, because input was moved to FEDIT `bandplot menu`
`=.kp`	defines, at which k-points the bandstructure is computed
`=.coeff`	if this file exists, the wave function coefficients for the reduced valence problem $$HC = SCE$$ are written to `+coeff`.
`=.atcharge`	defines the (non integer) nuclear charge for selected atoms (Wyckoff positions)

`=.sym`, `=.in` and `=.basis` are handeld by FEDIT , it is **no recommended** to manipulate them manually

[2] There may be more class-1 files, which are not documented here. However, the normal user is not expected to need them. There are some utility programs for the package, which use the same file name convention and thus have additional class-1 files (e.g. BANDPLOT). These are also not documented here.

=.sym contains the crystal symmetry. From the information contained in this file =.in and =.basis are recalculated. Recalculation happens if =.in is absent or if the status flag in =.sym requires an update. Normally this is done by FEDIT. On update, the non default settings of the existing =.in will be retained unless the symmetry in =.sym contradicts these settings, in which case the default settings are used. The code will notify these reset-events after update[3].

In a standard FPLO calculation run (no update action) the symmetry settings of =.in are used even if =.sym contains a different symmetry. Normally, if the update function of FEDIT was used, the symmetry settings of both files are equivalent.

If =.in exists and =.sym is absent, FPLO will extract a valid =.sym from =.in (this is usually done while invoking FEDIT).

=.in contains the major control data for FPLO. You can manipulate it interactively with FEDIT or automatically with the FEDIT-pipe-mode.

=.basis contains the basis definitions. This includes the definition which orbitals enter the calculation (core/ semi-core/ valence/ polarization) and the inital definition of the compression parameters (x_0) of the confining potential. The file content may be manipulated using FEDIT.

In auto optimizing mode (option BASIS_OPTIMIZATION) the file is updated with the new value of x_0 while FPLO is running. So, be aware if FPLO is running and FEDIT is used meanwhile, that the file content may have changed on disk during the FEDIT session. You will be promted to overwrite =.basis on exit (quit/save) in case that the content changed, while the FEDIT session was active. Usually it is correct **not** to overwrite the =.basis content in such a situation.

For all these files, read the help screens of FEDIT!

=.dens contains the density contributions of all sites (in terms of radial functions, which are the coefficients of the angular momentum expansion of the **overlapping** site densities). This file serves as input and ouptut for FPLO. It is created by means of a simple atom-like calculation on FPLO startup, if not yet existing. The density file may be re-used in other calculations (often a better starting point than the atom densities) if the number of sites in the unit cell and the type of atoms are equivalent.

=.densgrid To create real space density or potential plots create this file. It contains header informations and grid data. It may contain comments (line starting with '#') everywhere. It may contain empty lines.

[3] When FEDIT is used, see the protocol screen after symmtry update.

Comments at the top of the file (before data lines) may contain control settings. All other lines contain real space vectors in cartesion coordinates. For each vector the density/potential is plotted into the file `+densgrid`.

Header:

Important: please **no** tab-characters, only space!
All header control data are optional. Below we give the complete list of possible controls and their options. Any combination and order of options is valid. Options must be separated by at least a single space. The colon after the control key-word (`output`,`data`,...) may be separated with spaces. Any of the control lines described in 1-3 may be omitted, which forces default behaviour.

1. `# output: comments emptylines emptylines_with_space stop`

`comments`	output all lines starting with any space followed by '#'. If '`# output`' is the first line, it controls the output of all lines following, including itself.
`emptylines`	copy all empty lines of `=.densgrid` into `+densgrid`
`emptylines_with_space`	put a line with one single space into `+densgrid` for every empty line of `=.densgrid` (some plotting programs need this to separate data sets)
`stop`	stop after plotting, please be aware that you need a converged calculation to get reasonable densities/potentials!

   `emptylines_with_space` wins over `emptylines`, if both are specified!

2. `# data : index point total spin spinup spindown`

`index`	print index of grid point in first line
`point`	print grid point
`total`	print total density/potential
`spin`	print spin density/magnetic field (i.e. $n^{\uparrow} - n^{\downarrow}$)
`spinup`	print majority density/potential
`spindown`	print minority density/potential

3. `# type : density potential`

`density`	plot density data
`potential`	plot total-potential data

The last `type` specified, will win.

Plotting is done at the beginning of each iteration cycle after the potential is calculated.

Default:

1. `type:` density
2. `output:` no comments, no empty lines no stop
3. `data:` point total spin spinup spindown

The order of output data is fixed. So the `data`-options may be given in any order, but the output in `+densgrid` allways has the order: index point total spin spinup spindown. Of course, some of the fields may be absent, if omitted in '`data: ...`'

Remark: If some grid point falls onto an atomic position, the onsite contribution of this atom is neglected, since relativistic densities diverge at the nucleus ($s^{1/2}$ and $p^{1/2}$ orbitals diverge). Same holds for the potential (relativistic or not).

`=.ldos` **Obsolete since version 4.01**, because input was moved to FEDIT `bandplot menu`

To create *lm*-resolved densities of states, create this file in the directory, where the calculation is done. If FPLO finds this file the local orbital-resolved DOS is calculated.

Format:

line 1: number of lines following

line 2,...: number of site, for which a resolved DOS should be created

'sites' means all sites in the cell, not the Wyckoff positions! The definition of sites is found in the FPLO-output section '`UNIT CELL CREATION`', part '`Atom sites`'.

`=.kp` If this file is found by FPLO, the bandstructure (written to `+band`) is calculated at the points given in `=.kp`. The points are given in units $\frac{2\pi}{a_0}$, thus i.e. for bcc lattices the line Gamma-H consists of all points between $(0,0,0)$ and $(1,0,0)$.

Format:

line 1: [number of *k*-points] [only partially occupied bands] [lower offset] [upper offset]

line 2,...: one *k*-point per line (3 real numbers, separated by space)

Explanations:

The three entries behind the number of *k*-points in the first line are optional. They are used by the newer versions of the program `xfsf` to reduce file size.

[only partially occupied bands] a logical (t/f). If this is t only partially occupied bands are written to the files +band or +bweights. This reduces file sizes considerably, especially when used in conjunction with the Fermi surface program xfsf.

[lower offset] Additionally that number of bands below the lowest partially occupied band are written to the files.

[upper offset] Additionally that number of bands above the highest partially occupied band are written to the files.

=.coeff If this file exists, the coefficients of the reduced valence problem

$$HC = SCE$$

are written to the file +coeff. The file =.coeff may be empty.

Attention: The full wave function is constructed from this information **and** from the core and core-valence contributions, which are not contained in +coeff.

=.atcharge Somebody may want to use the virtual crystal approximation (VCA), which consists basically of introducing non-integer nuclear charges. This may be done by defining the content of this file.

Format:

line 1: number of lines following

line 2,...: number of Wyckoff-position followed by the nuclear charge

3.3 File class 2

The prefix '+' indicates the primary use as output from FPLO. Nevertheless, the files are partially input files for subsequent runs or tests. (Examples: debug files like +pot, +loi, +loitest)

+dos.total This and all other DOS files (except the +ldos files) are created if the option "CALC_DOS" is set or if "Bandstructure plot" in the bandplot menu is true.[4] It contains the total density of states.

[4] In CPA calculations there is no "bandstructure plot" option. Instead the Bloch spectral density may be calculated.

+dos.total.l contains the *l*-projection of the total density of states. E.g. the *d*-DOS is the sum of all *d*-orbital contributions of all atoms to the total DOS. See comments inside the files to see which angular momentum *l* is contained in the file. The numbers in the file suffix are just counters.

+dos.s contains the sort projected DOS. This is the sum of all DOS contributions of all sites belonging to the same Wyckoff position (sort).

+dos.s.l contains the sort and *l*-projected DOS. This is the sum of all *l*-DOS contributions of all sites belonging to the same Wyckoff position. See comments in the file.

+idos. These are like the +dos.-files but with the integrated DOS.

+ldos.s.l These files are created if required (see page 11) and contain the site and *l*,*m*-resolved DOS. The file numbers are running indices, the real site,*l* and *m* numbers are written in comments inside the files.

+error Created (and updated), while FPLO is running. It contains a summary of warnings and error messages. The update mechanism does not work on some platforms. Thus at the moment, it is best practice to check the FPLO output, since all messages are duplicated to standard output.

A lot of messages contain the number of the iteration step, where they occured. Normally, only the messages of the last iteration step (before convergence) are relevant.

Example: the notification, that core states have to be treated as semi core states.

+run contains the hostname and the PID of the last run of FPLO. If it is still running, this information may be used to kill the job.

Caution: Killing FPLO, may cause loss of the =.dens file and therefore loss of the calculation result, in case that FPLO is just writing the file =.dens, when it is killed! The same holds for =.basis.

+band, **+bweights** are created if the band-structure/band-weights plotting options have been set via FEDIT. They contain the band-structure/band-weights. Use FEDIT -bandplot[5] to produce the related pictures from them.

---

[5]This will use BANDPLOT to create a Postscript file.

+points Created in the initialization phase at the beginning of the FPLO run. It contains the special symmetry points used for the band structure creation. It is used by BANDPLOT.

+voronoi Created in the initialization phase at the beginning of the FPLO run. Contains the voronoi cell geometry.

If 'build voronoi' is true, +voronoi is created and used. If 'build voronoi' is false, one may specify 'voronoi file' as an alternative file to read from. This serves for cell merging. However, cell merging is not expected to be needed in versions later than 3.00.

+symmetry Created by the symmetry module of FPLO. It contains information about the crystal symmetry.

+symanalysis Created by the symmetry module of FPLO. It contains information about the crystal symmetry induced conditions for the (non relativistic) matrix elements of the onsite blocks of the density-matrix/Hamiltonian/overlap-matrix. The diagonal elements give the conditions for the site and angular momentum resolved DOS.

+fcor.sort.spin These files are created, if option PLOT_BASIS is switched on. They contain the radial part of the core states.

+fval.sort.spin These files are created, if option PLOT_BASIS is switched on. They contain the radial part of the valence states.

+fdval.sort.spin These files are created, if option PLOT_BASIS is switched on. They contain the radial part of the derivative of the valence states with respect to the prefactor (λ) of the confining potential

$$V^{\mathrm{cf}} = \lambda \left(\left(\frac{r_{\mathrm{NN}}}{2} \right)^{\frac{3}{2}} \right)^{-4} r^4$$

$$\lambda = x_0^{-4}$$

+atcor.sort.spin, +atval.sort.spin Created if option PLOT_REALFUNC is switched on. Contains the effective potential, used in the Hamiltonian of the radial atom like equation for the calculation of the basis orbitals

$$V = \frac{l(l+1)}{2r^2} + \left\langle \left\langle V^{\text{cryst.}} \right\rangle_{\text{spherical}} \right\rangle_{\text{smoothed}}$$

$\langle V^{\text{cryst.}} \rangle_{\text{spherical}}$ is the spherical average of the crystal potential around the considered site. The potential actually used (and contained in the files) is a smoothed version of V!

+pot.site.spin, +dens.site.spin, +vxc.site.spin, +har.site.spin, +ewp.site.spin, +vat.sort.spin Created if option PLOT_REALFUNC is switched on. All these files contain the angular momentum components. They contain

- The potential contributions of site 'site'.
  The $L = 0$ part is multiplied with $\frac{r}{\sqrt{4\pi}}$. So, it starts at $-Z$ (nuclear charge) at the origin.
- The density contributions of site 'site'.
  The $L = 0$ part is multiplied with $\sqrt{4\pi}r^2$. So, it integrates to the electron number belonging to the respective site.
- The (shaped) vxc potential contributions of site 'site'.
- The hartree potential contributions of site 'site'.
- The (shaped) extended ewald potential contributions of site 'site'.
- The offsite contributions to the smoothed spherically averaged crystal potential around Wyckoff position 'sort' (excluding the contribution from the site itself). The spherically averaged potential (sum of onsite and offsite of all sites) has a maximum near $\frac{r_{\text{NN}}}{2}$. Between this maximum and the maximum radial mesh point the spherically averaged potential is interpolated/smoothed such that the asymptotic smoothed potential equals the totally averaged crystal potential[6]. The file +vat... contains the difference between the smoothed potential and the local site potential. (The $L = 0$ part of +pot... devided by r plus +vat... gives the $L = 0$ part of +atval...)

Be aware, that the local contributions have no physical meaning, only the sum of all has one.

+fedit, +fedithelp Communication files between the executables FPLO/ BANDPLOT/ DIRAC and FEDIT. These files are created on every run of the executables. In the initialization sequence of the FEDIT run, they are deleted and the appropriate executable is called, to recreate them. In this way it is assured, that FEDIT allways reads the correct information.

---

[6] There are various smoothing algorithms. One of them does not fulfill this condition. However, the default algorithm does.

+dirsh Created, if shape test is performed. It contains the shape function along the lines specified in the shape sub menu of FEDIT.

+unity Created, if shape test is performed. It contains the sum of all shape functions minus 1 along the lines specified in the shape sub menu of FEDIT.

It should contain zeros, at least near the origin. (For larger distances the summation of all shape functions may be incomplete.)

+densgrid See =.densgrid

3.4 File class 3

These files have no prefix.

---

bravais.ps, **primitive.ps** Created in the initial part of the FPLO run, before the density is read. Contains a postscript picture of the bravais/primitive cell.

---

vcell{site}.ps Created in the initial part of the FPLO run, before the density is read. Contains a postscript picture of the voronoi cell of site 'site'.

3.5 Directories

FEDIT creates/uses a subdirectory +tmp in the directory where it was called, to perform input file updates. This directory may be deleted after use of FEDIT (or at the end of the calculations)[7].

[7] Sombody will argue that we should use the systems tmp directory. May be. But our way of doing it is independend of the system settings.

Chapter 4

Automation, scripting, pipe-mode

The manipulation of input files via unix commands like `ed`, `sed`, `awk` and similar ones is strongly discouraged. The format of the input files follows a syntax and is not fixed in position. To achieve automation the pipe-mode of FEDIT can be used.

4.1 Rules

There are only a few rules to be obeyed.

1. The menus/screens/edit-controls of FEDIT are operated by ascii hotkeys in interactive mode. For some actions there are control keys (cursor movement, searching and scrolling) used. In pipe mode the latter keys are not needed at all. The ascii hotkeys are replaced by a simple syntax explained below.

2. In interactive mode sometimes only a portion of the menu's form is seen on screen. In pipe-mode allways the full form is virtually visible. There is no need for scrolling. (In fact there is no need for scrolling in interactive mode as well, since the currently invisible hotkeys are also active and just typing an currently invisible hotkey will scroll the form to make the selected edit-control visible.)

3. In interactive mode the edit-controls may be edited to change only part of the controls content. In pipe-mode the full content/data of the edit-control has to be entered.

4. In interactive mode there are toggling actions, in pipe mode these become normal edit-controls.

5. In interactive mode there will be informational screens displayed after certain actions, which just show output. In pipe-mode these screens will not occure. (E.g. the FPLO message after symmetry update.)

6. In interactive mode there will be questiones to be answered depending on the context. In pipe-mode such questions naturally cannot occure.

7. In interactive mode, the user enters information by typing keys, in pipe-mode the user input to FEDIT is read from standard input (`stdin`). This input may e.g. be stored in a file or come from a `here-script` within a shell/perl-script.

8. In interactive mode the user feedback is what is seen on the screens, in pipe-mode the feedback is written to standard error (`stderr`). So it is good practice to redirect `stderr` to a file and to redirect standard output to `/dev/null`. (The screens on `stdout` will change so rapidly that they are useless.) Let us assume that the input for the pipe-mode is stored in the file `=.pipe`. (The prefix indicates that it is an essential input file.) A good way to feed the information contained in `=.pipe` into FEDIT in a shell environment would be

   ```
   cat ./=.pipe | FEDIT  -pipe 2>./+log 1>/dev/null
   ```

   or

   ```
   FEDIT  -pipe < ./=.pipe 2>./+log 1>/dev/null
   ```

   Both commands have the same effect.

   (a) The first command uses `cat` to write the content of `=.pipe` to `stdout`, which than is redirected to the `stdin` of FEDIT via the unix pipe command `|`. (That is the origin of the name pipe-mode.) The second command uses the unix tool for redirecting the `stdin`. Here the content of `=.pipe` is directly written to the `stdin` of FEDIT.

   (b) The editor is given the option `-pipe` to setup the mode.

   (c) The `stderr` is redirected to the file `./+log`. (The naming is up to the user, however, our choice follows the rules explained in Chapter 3. The '`+`' indicates that the file is not essential and may be deleted after a succesful run.)

   (d) The `stdout` is redirected to the unix device `/dev/null`, which just means to discard it. (`/dev/null` is a 100% information sink :-)

9. The return code of FEDIT should be checked and the script aborted if needed, to intercept input errors. FEDIT sets the shell exit/return code as

`1`	on success
other	on error

   The exit/return code must be checked imediately after the command, which returned it. Any shell action in between changes the exit/return code!

10. The logic of the pipe-mode is such that the user has to feed hotkey-data sequences and menu-hotkey sequences into FEDIT as if he would in interactive mode. The user creates key sequences

in the pipe-input, which navigate through the menus as if it would be an interactive session. This includes the `x`-hotkey to leave a sub menu. It excludes the `x`-hotkeys of information screens as described above. (The main reason is, that those screens are context dependend.) If there is any invalid input in the pipe file, the editor will abort unsuccesfully. The reason is printed to `stderr`, which is accesible as explained above. To create pipe input, just use the editor interactively and write down or remember all hotkeys, which are pressed to complete the desired editing (except the information screens and questions).

Altogether the simple rule is that in pipe-mode there is only navigation to certain positions and entering of data. Every other user $\Leftrightarrow$ FEDIT interaction will be absent.

4.2 Syntax

We assume the pipe input to be stored in a file. Its syntax is as follows.

1. The file may contain comments, which are lines whose first non-blank character is '#'.
2. The file may contain empty/blank lines.
3. Every single action goes into a separate line.
4. A hotkey is wrapped into two `'@'` characters.
5. An alternate sub menu ('<SPACE>hotkey') is called with a single space followed by the hotkey, wrapped into two '@' characters
   ```
   @ hotkey@
   ```
6. A hotkey sequence, which activates an edit-control to enter data is mapped onto a line
   ```
   @hotkey@data
   ```
7. A hotkey sequence, which activates *N* edit-controls is mapped onto
   ```
   @hotkey@data1@data2@...@dataN
   ```
   Example: The basis menu contains a basis definition and a compression edit-control for every Wyckoff position. The hotkey equals the sort number. A corresponding pipe input to edit the second sort could look like
   ```
   @2@ 1s : ( 2s2p) / 3s3p3d + 4d @ -1 * * * 0.9
   ```
   This sets the basis definition including all delimiters and the compression paramter for sort 2. The sign '*' is a placeholder for the value, which was there before. Usually one uses a well defined set of input files (created, e.g., by an interactive FEDIT session) as starting base for the pipe-mode. The file `=.basis` thus would already contain the proper basis. The placeholders avoid overwriting of self consistently determined x_0 values from the previous run.

Consider a case where a set of converged calculations has to be restarted to create additional output (say bandstructure). Now, one does not want to destroy the converged x_0, while at the same time using the same pipe-files as in the previous run (where input was created first time). The only change to the pipe file in this example would be to switch-on the `bandstructure plot` option. The placeholders serve this goal. The fixed x_0 are given explicitly but the free x_0 are untouched.
Some remarks on the basis set syntax. The string must have the form as it is displayed in interactive mode. It has a core, semi-core, (free) valence and polarization section. The sections are devided by the delimiters ':', '/' and '+'. All three delimiters have to be used. (Even if the polarization section is empty, we need the final '+'.) Blank characters do not matter between the single input units. Units are delimiters, orbitals and parentheses. The delimiters may be repeated any number of times, so a string '`1s : :: (2s2p) / ...`' is permitable. (Imagine it as typing three times ':' at the same position in interactive mode.) The individual orbitals must be pairs of numbers and characters **without** space in between.
The number of compression parameters must fit the basis definition. The x_0-paramter may be entered with any sign, since the sign is allways automatically adjusted to be negative for semi-core and polarization orbtials and to be positive for (free) valence orbitals. However, to have a clear syntax it is better to enter the minus sign for the fixed x_0 explicitely. The '`*`'-placeholder should be used for the free orbitals, except for cases where an explicit reset to a certain starting value should be achieved. Make sure in such cases, that you replace these x_0 values by '`*`' after the reset took effect (script was executed and input files were changed), to avoid destruction of self consistent x_0 on later calls to the script!!!

8. An edit-control may be activated by a search command (as in interactive mode). This is done by
   `!search-string!data`
   This will look for the first occurence of the search string in the current form and if this markes an editable control, it is activated and data is used as the control's new data. Multiple edit-controls, selected by search are mapped onto
   `!search-string!data1@data2@...@dataN`
   (Note: As in the previous point, the '`@`' replaces the '`<ENTER>`' key used in interactive mode.)

9. An edit-control, which is toggling in interactive mode is not toggling in pipe mode. So, if such data shall be edited in pipe mode one needs to set the value explicetly. This assures a defined state of the input after completion.

   (a) logical values may be '`t`' or '`f`'

   (b) binary values take there values as they are written on screen in interactive mode. (Example: spin sorts may be '`1`' or '`2`')

   (c) Options (marked as selected by `[X]` on screen) are selected or deselected with the values '`+`' or '`-`' in pipe mode

Example:
To select the option `NO_SYMMETRYTEST` in the options submenu we may use the search tool

```
!NO_SYMMETRYTEST!+
```

or to deselect it use

```
!NO_SYMMETRYTEST!-
```

The search string must be choosen to be unique in the way that it really selects what you want. It is best used only in options menus and in select boxes.
It is strongly discouraged to activate options (the things marked with '`[X]`') via the related hotkey, since these hotkeys are created automatically by FEDIT **and thus may change if the version changes**[1]**. The search utility is designed exactly for the purpose of changing options.** In select boxes the hotkeys will not change. However, the search utility gives better readable pipe-files.

[1] In fact, there is a list of all possible options. FEDIT takes this list and creates hotkeys for all of them. If the order of options in this list changes between different FPLO versions then the mapping of hotkeys to options will naturally change as well. The authors try to avoid a change of the options order. However, there are thinkable cases when this will be nessecary!

10. The alternate choices in a select box are selected by a single hotkey command as in

```
...
# we assume we are in the main menu here
# we enter the relativistic select box (hotkey 'r' in the main menu)
@r@
# we select the full relativistic mode, by searching for
# 'full relat', which is uniqe in this select box
!full relat!
# We equally could use the hotkey, off course.
# Let us select it again:
@f@
# Now we leave the select box and go back to the main menu
@x@
# continue
....
```

Please observe that the parentheses indicating the hotkey (`'(F)ull relativistic`' in this case) are not visible in search mode, interactive or not.

11. The last key sequence must be

```
@q@
```

and must be called in the main menu. (So, you need to virtually navigate back to the main menu.) This is to assure proper cleanup.

4.3 How to set up automation

There is one pecularity. Suppose you created a series of input files and converged all related calculations. Now, you want to change some settings, for instance increase the number of *k*-points, since it turned out that you had to few of them. You could do this with a minimal script, which only changes the number of subdivisions in *k*-space. Later you recognize that you need to change other settings. Again you could do it with a minimal script. The main drawback of this approach is, that you loose control over the changes, which have been done. Another point is that after a change of certain symmetry parameters some input data are reset to default values. For these reasons it is advisible, to set up the pipe file(s) always such that the input files are completely determined by the content of the pipe file.

If FEDIT is called within a directory containing input files it offers the content of these files for editing. This means that the state of input files depends on the previous editing. The following sequence of actions will assure a state that does not depend on previous editing.

1. Go to the symmetry menu and enter all symmetry parameters explicitely. (These are not so many data, so it is not a big task.)

2. Call `update` with the '+' hotkey! This will reset certain data of the `=.in` and `=.basis` files. **Never forget this point!** Most of the settings will remain unaltered. All calculations of a series should belong to the same symmetry type. That means, the Wyckoff positions should at most change their parameters but the spacegroup should stay the same.

3. Go back to the main menu (hotkey 'x')

4. Call the alternate submenu action `Recreate` (hotkey '<SPACE>e'). This will reset the file `=.in` to its default values, which belong to the symmetry setup in `=.sym`. Remember that in pipe mode the corresponding question (`Really rebuild default content ? (y/n):`) will not be asked, so you must not type 'y' or 'n' in pipe mode. Following the answer 'y' an informational screen will appear in interactive mode, which will be left with 'x'. This screen will be absent in pipe mode, due to the general rules above. That means that in interactive mode you would type the sequence '<SPACE>eyx' to perform the rebuild, but in pipe mode the action is achieved with a single line

   ```
   # assume we are in main menu
   # next line will call the reset action
   @ e@
   # we are back in main menu now
   ```

5. Now, enter all input data except symmetry data, which differ from default input.

A setup for a series of calculations may be done as follows.

1. Create interactively an initial input for your compound. Perform a self consistent calculation. Check if everything is fine. Converge the number of k-points (and Fourier components, if automatic mode is not used) (may already be done with the help of scripts, off course).

2. Setup the scripts, which create the input for the series. Make sure to perform every calculation in a seperate directory! After creating a new directory for a particular calculation of the series and before invoking FEDIT copy `=.sym`, `=.basis` and `=.dens` from the initial calculation directory into the new directory. This has two advantages. First, the preconverged x_0 parameters (from the inital calculation) will be available in the newly created input and second, the preconverged density will be available. As a result convergence will be much faster (in many cases) than in a calculation from scratch. This procedure makes sense as long as the parameter variation in the series is not to large. But even if so, the proposed procedure will not hurt.
   (In calculation series, (e.g. search of the minimum of an energy surface using e.g. a steepest decent algorithm) where the input paramters depend on previous calculations, one may copy one of the latest previous calculations into the new directory instead using the inital calculation.)
   Make sure that the '`=.`'-files are copied only, if a new directory has been created. (Do not over-

write converged basis or density.)[2]. If the script is re-run with some paramters changed, the basis and density input of the previous self consistent calculation should be retained!

3. Run the script to create the directories with the proper input. Check the created input interactively!

4. Launch all calculations in the various directories and wait for self consistence. Redirect output into a file, best is to allways use the same file name (for instance `'out'`). Check from time to time if everything runs fine.
   You may modify your script such that it serves the input creation and the starting of the calculations. Or you create a separate script to launch the calculations. Use the specifics of your platform (may be there is a job queue).

5. Check if all calculations realy converged. There is a final message on completion of a calculation[3]. Check this. The final message has the structure
   ```
   TERMINATION: Keyword : explanation
   ```
   where keyword may be

`Finished`	The calculation run to self consistence or a single step calculation finished.
`Normal`	The calculation did what it was ment to do and achieved the goal. Some actions terminate the program before self consistency, and this triggers a normal termination.
`Error`	Something happend. In many cases there is a cure for the problem.
`Crash`	This is really a bad case.

   The Keyword may be checked automatically with the help of unix tools. The explanation string gives a rough idea, what happend.

6. Have a look at example scripts in the distribution on how to extract certain data from the output.

7. Now, you are ready and may modify the script to change some settings and re-run the series if nessecary. Take care about all points mentioned so far. Especially, use the placeholder syntax for the x_0 data, to avoid messing up converged results. Example: increasing the number of k-points usually needs only a few steps to achieve convergence, if started from a preconverged calculation.

[2] This means that the script, which creates the directories and the input should test the existence of the directory.
[3] Except a bug occured.

4.4 Advanced features

4.4.1 Basis equivalence classes

There is the tool of basis equivalence classes for chemically very similar Wyckoff postitions. Some rules should be followed to achieve a secure automation. In interactive mode the equivalences are defined by typing `':'` in the `basis menu`. This opens a small edit-buffer, where two commands may be typed:

`eqm sort1 sort2 ... sortN` will move all $N-1$ sorts (given first) into the equivalence class of the N-th sort (given last), retaining the basis setting of the N-th sort. If some of the sorts to be moved are contained in another class, they are extracted from this class before moving them. On the screen the classes are displayed such that the sort with the smallest number in this class will determine the hotkey of the corresponding edit-controls. The target sort of a move command need not nessecarily be the first sort of the target class.

`eqx sort1 sort2 ... sortN` will extract the specified sorts from which ever class they are contained in and makes them single-membered classes.

In pipe-mode these commands are available, but there is a subtle point, which may lead to errors. The problem is the resorting of the sorts in a class, which reduces the number of available hotkeys in the basis menu in a way, which may be missed by the user. The safest way to avoid this error is to

1. extract all sorts from where ever they are.

2. edit the basis (at least the ones belonging to single membered classes and the ones which are target sorts in the following move command, but better define all sorts.)

3. form classes by moving

To give an example lets suppose we have a compound with 8 Wyckoff positions with 3 very equivalent Fe atoms, which are to be treated within the same basis class. The sort numbers of these three Fe atoms shall be 2, 3 and 7, say. A clean way to create the basis input would be

```
# suppose, we are in the basis menu
# first extract all sorts
@:@ eqx 1 2 3 4 5 6 7 8
# define basis for all single-membered classes and for the Fe class target
# sort (number 3 in this example)
@1@ ...
# sort 2 is skiped since it is moved to three later
@3@ ...
@4@ ...
@5@ ...
@6@ ...
# sort 7 is skiped since it is moved to three later
@8@ ...
# Now make the class. It is important that the target sort (3) is defined
# above. If 3 is not the last (target) sort  in the next line, than an
# undefined state may result, since sort 2 and 7 were not defined
# explicitely above!!!
@:@eqm 2 7 3
```

4.4.2 Initial polarization, inital spin split

Some times, you may encounter a situation where it is nessecary to repeat an initial spin split for a series of calculations, since for example, the calculations were started spin polarized with an insufficient split and run back to a non magnetic state. Instead of discarding all calculations done so far, it is better to force a new inital split on the preconverged density and basis. But, we face the situation that there are already 2 spin sorts in the density files and thus another `inital polarization` would urge FPLO to pose the question

```
Do you want to skip the possibly repeated initial spin splitting ?
```

Normally, the answer would be '`y`' since mostly one just forgot to switch off the `inital polarization`. But in our situation the answer would be '`n`', which forces another split. (What we really want is to restart the calculations in a new attractor basin.)

Now the problem is that usually the FPLO jobs are started in a background mode. In such a case there is no user to answer the question and so the job will crash (due to the fact that it tries to read from `stdin`, which is not present in background mode). If the job is not running in background it will hang and wait for the answer. Imagine, you started the script before weekend and come back at monday and it still is waiting for the answer, or crashed.

There is a way to circumvent the problem: One provides FPLO with the nessecary input on `stdin`. If the program is not reading from `stdin` (since the question did not arise) the `stdin` will just be ignored and

no problems occur. But if the question arises the proper answer is available and the program continues. There is an easy way to do this.

In the script which actually lauches the job there will be a line like

```
FPLO 2>/dev/null >out
```

We have to change it into

```
FPLO < ./+yes 2>/dev/null >out
```

or

```
cat ./+yes | FPLO 2>/dev/null >out
```

where the file `+yes` has to be created before (by the script) in the directory, where FPLO is running. This file contains a single line with the proper answer ('`y`' to skip the split in allmost all cases or '`n`' in the particular situation, which we discussed above.). The file may be created with a `here-script` like in

```
cat <<EOF > ./+yes
n
EOF
```

or with `echo` like in

```
echo "y" > ./+yes
```

So, comming back to our situation. To enforce the repeated spin split we would (in the script) switch on the `inital polarization` and we would put an '`n`' into the file `+yes`. After a succesful split (means after the re-started caclulations passed the splitting), we should imediatly replace the '`n`' by an '`y`' and and switch off the `inital polarization` in the script, to avoid an unwanted destruction of the now hopefully correctly converged results.

In fact one should allways use this construct, just in case, that one forgot to switch off the `inital polarization`.

Some people will know the unix command `yes`, which will do a similar job as the file construct. However, in practical applications we often met the situation, where this command was not available in a job-queue environment. Therefore, we decided to use the file trick.

4.5 Example

We give an example script here, which uses basic features explained above. It is contained in the distribution directory (in case of default installation something like FPLO/FPLO5.00-18) under DOC/Scripting_example/.

```
#! /bin/sh
#
# Example script to create a series of calculations for varying lattice
# constant for fcc Al.
#
# We use the so called here-script mechanism to create the pipe input
# on the fly.
# This script works with bash at least. For other shells part of the syntax
# may be different. Consult your man pages.
#
# We assume that the script is exectuted in a directory, where there is
# an inital calculation sub directory called SC containing a converged
# calculation of the same compound (fcc Al).
#
##############################################################################


# Always use fully qualified names, to assure proper program version.
# Store the exec names in variables, so we do not need to scan the script
# to replace the version latter on.
FEDIT=fedit5.00-18
FPLO=fplo5.00-18


# Give all directories a name prefixed with the name of the parameter,
# which is running, followed by the parameter itself
prefix='a0='

# Remember the current directory.
# Be aware of the back quote syntax, on some unix systems you need a different
# construct to cast the output of a command (pwd here) to a string.
ROOT=`pwd`


##############################################################################
```

```
# some functions
usage() {
  echo "usage:  $1 [-r]  [-h[elp]]"
}

#########################################################################

# Check the command line flags.
#
RUN_IT=0

while :
  do case $1 in
    -r) RUN_IT=1
        shift 1
        continue
        ;;
    -h*)
        usage ‘basename $0‘
        exit
        ;;
    -*) usage ‘basename $0‘
        echo ’wrong options’
        exit
        ;;
    *)  break
        ;;
  esac
done


#
# security questions
#
if [ x$RUN_IT = x1 ]
then
    echo "Shall I run the jobs: [y/n]" ; read YN
    # The next test is a little trick to circumvent problems with
    # empty variable. We first form the concatenation x$YN, which
    # has the value of $YN prefixed with a single x.
```

```
    # Then we compare it with the teststring (y in our case) prefixed by x.
    # So, if $YN=y than also x$YN=xy. Disturbing?
    if [ "x$YN" != "xy" ] ; then
        echo "abort"
        exit
    else
        echo "Will run jobs now."
    fi
else
    echo "Shall I (re)create the input: [y/n]" ; read YN
    if [ "x$YN" != "xy" ] ; then
        echo "abort"
        exit
    else
        echo "Will (re)create input  now."
    fi
fi

#############################################################################
```

```
# loop over the running parameter, in our case the lattice constant
for xx in 6.00 6.50 7.00 7.50 8.00 8.50
do

    # make sure we are in the root directory of our data directory tree
    cd $ROOT

    # create the directory name as described above (example 'a0=6.00')
    dir="$prefix$xx"


    # check, if input creation or job running shall take place
    if [ x$RUN_IT = x0 ] ; then
        # input creation branch
```

```
        # if the directory does not yet exist, create one
        if [ ! -d $dir ]
        then
            mkdir $dir
            echo "directory $dir created"
            # copy the essential files from the inital calculation
            # which is assumed to be in the directory SC
            cp ./SC/=.basis ./SC/=.sym ./SC/=.dens $dir
            echo " =. files copied"
        else
            echo "directory $dir exists allready"
        fi


        # change into the directory of paramter $xx
        cd $dir

        # Now create the pipe file content, with the help of a here-script.
        # For the sake of book keeping we will save the pipe info into a file.
        # (One could equally pipe directly into fedit.)



        # The next command is the here-script ( <<EOF ), whose stdout is
        # redirected ( > ) into the file ./=.pipe.
        # Every thing which comes between the <<EOF line and the line below,
        # starting with EOF, will go into ./=.pipe. The main advantage of this
        # approach is, that we may use shell variable replacement
        # (interpolation) to put the information of the running variable
        # $xx at the proper position
        cat <<EOF  > ./=.pipe
###############################################################################
# this is the beginning of the pipe file

# go to symmetry menu
@+@
    # title
    @c@Al, a0-variation
    # enter spacegroup select box
```

```
    @s@
        # select space group
        @225@
        # leave selectbox
        @x@
    # lenth units
    @u@
        # bohr radii
        @b@
        # leave select box
        @x@
    # lattice constants; Here we put our running variable.
    # We enter as first lattice constant the value which is in $xx,
    # then we use the fedit-',’-syntax to repeat the value
    @l@ $xx , ,
    # set axis angles
    @a@90.,,
    # usually you will not change the "Maximum L", but it does not hurt
    # to have it here
    @m@12
    # setup Wyckoff positions
    # number is one in our case
    @n@1
    # Now, give list of ALL !!! Wyckoff positions.
    @1@ Al @ 0.,,
    #
    # NOW CALL UPDATE, NEVER FORGET THIS!!!
    #
    @+@
    # leave symmetry menu
    @x@
# back in main menu
# This was the symmetry setup, and now we follow our advise to create
# the default =.in input by using the REBUILD-action.
# (The space before the ’e’ opens the alternative menu bar.)
@ e@
# now we have the default input, and are still in the main menu

# Let us set the number of k-points to a non default value:
@k@ 16,,
```

```
# Let us set the xc-potential version now:
# first enter select box
@v@
    # select via search
    !Perdew Wang 92!
    # leave select box, go back to main menu
    @x@

#Let us set the relativistic mode:
@r@
    # Select by search, please note that the parentheses indicating the hotkey
    # are not considered in search mode.
    # (Have a look at the select box interactively.)
    !scalar relativistic!
    # leave select box
    @x@



# Let us set some options:
# enter options menu
@-@
    # Good habit is to select all options explicitly
    # This includes the options, which are selected by default.

    # here an example for deselecting
    !BASIS_OPTIMIZATION!-
    # here an example for selecting,
    #          (off course, we want to optimize the basis)
    !BASIS_OPTIMIZATION!+
    !PLOT_BASIS!+
    !NO_SYMMETRYTEST!+

    #leave menu
    @x@


# Let us select spin=1 explicitly:
@s@1
```

```
# Let us switch off inital polarization explicitly:
@i@f



# enter basis menu
@b@
    # Set the basis of all Wyckoff positions explicitly  !!!
    # Here we have only one position.
        @1@ 1s : (2s2p) / 3s3p3d + @ -1 * * *
    # leave basis menu
    @x@



# last action must be
@q@

# this is the end of the pipe file
###############################################################################
EOF


        # Now execute fedit in pipe mode and use the information of the file
        # =.pipe just created in $dir.
        # We made sure that we are in $dir, since fedit shall act there!
        # We explicitely tell fedit to use the fplo executable stored in the
        # variable $FPLO to have a well-defined state.

        $FEDIT -p $FPLO -pipe <./=.pipe 2>./+log 1>/dev/null

        # Check exit/return code of fedit, there must not be any command
        # in between the check and the command, which produced it (fedit here).
        # The return code is stored in the variable $? in shell.
        # exit=1 means success
        if [ $? -ne 1 ]
        then
            cat<<EOF
Content of log file:
```

```
------------------------------------------------------------------------
EOF

            cat ./+log

            cat <<EOF
------------------------------------------------------------------------
There was an error in the pipe input. Check logfile above or in $dir/+log.
Be aware that the line numbers refere to the file $dir/=.pipe!
EOF
            exit 2;
        fi

        # If we are here, the input was created in $dir according to our setup
        # Now we continue with the next parameter

        # change back to where we started
        cd $ROOT


        # end of input branch

    else

        # job-run branch

        # change into the directory of paramter $xx
        cd $dir

        echo "$FPLO running in  $dir ..."

        # now execute, whatever is nessecary to launch job in the current
        # directory (name $dir)

        #START: example
        # We just run the jobs sequentially on a single machine
        # and redirect stdout to file out and stderr to /dev/null.
        # (In this way there will be no dangling output and the job could run
        # savely in background, which is not done in our example here.)
```

```
        # Furthermore, we use the +yes-file mechanism to avoid a crash
        # due to repeated inital polarization (spin split).
        # The "y" below enforces fplo to continue in such situation
        # without a repeated split and  does nothing otherwise. See manual.

        echo "y" > ./+yes


        cat +yes | $FPLO 2>/dev/null > out

        #END:   example



        # change back to where we started
        cd $ROOT
    fi


# end of xx-loop
done



#
# After the input creation run we should have a directory structure like
#
# ./SC/
# ./a0=6.00/
# ./a0=6.50/
# ./a0=7.00/
# ./a0=7.50/
# ./a0=8.00/
# ./a0=8.50/
# ./script
#
# where every directory contains the same setup, except for the
# lattice constant.
# We may now perform converged calculations (option -r) in all directories.
# If we want to change, say, the number of k-points, we edit this number
```

```
# in the pipe-section above, re-run that script to change the input and
# re-converge the calculations (option -r).
#
```

To test the script, copy the directory `Scripting_example` (including the subdirectory `SC`) to an apropriate place, change into the copy and edit the variables `FPLO` and `FEDIT` in `script` to point to the fully qualified executable names of your instalation. Make sure that the `PATH` is properly set, so that the executables may be called in your environmet.

Then execute

```
./script
```

and answer 'y'. The directory structure described at the end of the script should have been created now. Next execute

```
./script -r
```

and answer 'y'. Now the calculation is running in sequence. Wait for it to finish. Test convergence via

```
grit a0=
```

Check termination status via

```
grTE a0=
```

Collect total energies in file named '`e`' via

```
grEE a0= |tee e
```

Plot '`e`' if you want to.

The scripts '`gr..`' are part of the installation and should be available if your installation procedure succeded.

Appendix

A FAQ

Q: How can one use existing input files with different versions of FPLO?

A: Updating is simple, just call the newer FEDIT. It will update the input files. Then continue with the corresponding FPLO.
For downgrading see Chapter 2.

Q: I started FEDIT and got the message

```
error(ReadPCTable): Cannot find entry definition file
 '+fedit'!
 It should be in the local tmp directory '+tmp'!
 One can overwrite the location using option -ef filename.
 Possible reason:
        1) the executable '...fplo...' is not running correctly
           or just does not exist!
        ....
```

A: Make sure that the FPLO executable (of the same version as FEDIT) is within the `PATH` (shell environment variable). Another possible source of the error is that the directory or some of the relevant files are write protected. If the FPLO executable is really not running (some linked libraries not found), just run FPLO by hand on command line and see what happens. Be aware of the fact that FEDIT will automatically try to execute the FPLO executable using the fully qualified name (with the 'version-release'-suffix, like `fplo5.00-18`). It will not use the generic name `fplo`.

Q: How can I save memory?

A:

1. The number of `occupied bands` can be specified in the FEDIT main menu. Read the help screens for this variable. This does **not** work for CPA.
2. For large compounds the number of *k*-points in the Brillouin zone may be reduced. (Convergence with the number of *k*-points has always to be testet!)

Q: I want to make an antiferromagnetic calculation...

A: You will need two `spin sorts` and have to set the `initial polarization` to 't'. Then there is a submenu (press `<SPACE> i`), which allows to set the value of the `inital spinsplit`. Choose the same absolute value with opposite sign for the atoms, which are antiferromagnetically ordered. Set zero for atoms, which by symmetry must have zero moment. To assure zero total moment you may use the FSM method with a total `spin moment` of zero.
Note, that in the current implementation equivalent atoms with opposite magnetic moments have to be put on different Wyckoff positions. This is one of the rare occasions where the code does not make full use of symmetry.

Q: I want to plot the real space density or potential...

A: See description of file `=.densgrid` in Chapter 3. The output is performed after the potential calculation in each iteration cycle (otherwise the potential is not yet defined). Use a plot program of your choice to visualize the data.

Q: FPLO cannot allocate memory.

A: Ask the system administrator to increase the limits of stack, heap and data size of a user job. FPLO jobs like other band structure programs need a lot of memory. The memory consumption is proportional to the number of *k*-points in the irreducible Brillouin zone, which in turn usually is proportional to the inverse of the number of atoms. You should check the number of *k*-points you really need to solve the physical question. For huge cells sometimes one may drop polarization states (this results in reduced accuracy, off course.).
Use number of `occupied bands` to decrease memory usage!

Q: I rebuilt the code and gave the executables non default names. How can I run FEDIT with the new code.

A: Use an option:

```
FEDIT  -p name_of_your_executable
```

This will force FEDIT to work with the executable named after -p. The executable can be an FPLO, BANDPLOT or DIRAC executable. Off course, the executable must be accesible within the current `PATH`.

Q: How can I see the progress of the calculation?

A: Assumed that you re-directed the FPLO output into a file, use the unix command:

```
grep "last deviation" fplo_outfile
```

Q: I want to use `less`, `grep`, `vi` or other unix tools to work with the FPLO-files having a name '`+...`'. It does not work!

A: Many (new) unix tools will interprete the '`+`' sign as an option flag. To use these tools with '`+`'-files, specify `./+file` instead `+file` on command line!

Q: I started a calculation, including upper core states (semi-core) into the valence set. I use auto-x_0-variation and the calculation does not converge, the x_0 jump around.

A: Semi core states in the valence should in almost all cases be fixed with respect to x_0-variation (they may be grouped with well behaved valence orbitals as well).

1. set the compression parameter to -1 (appropriate value for almost all cases, although not neccessarily the best value)
2. **NO LONGER RECOMENDED:** If the semi-core state is really well localized, you may set x_0 to 0.0 (means no compression = infinite x_0)

Q: I started a calculation including higher states into the valence set (polarization states). The normal states like Fe-3d, which where stable before, are now expanding, producing instable results.

A:

1. Check if there are semi-core warnings in the output. If so, obey them!

   Example: Fe with the basis "`1s2s2p3s3p:: / 4s4p3d4d +`"
   The semi core states 3s3p (which are treated as core states here) do heavily overlap. This produces a conceptual error, which corrupts the assumptions for the optimization process. Include the 3s3p semi-core orbitals into the valence set (semi core section), with fixed x_0 or via grouping. It will work (altough a bit slower).

2. Check what happens to the polarization state. It almost certainly must be grouped with some other state, e.g., with a state with the same orbital momentum quantum number and next lower principal/main quantum number.

Q: I have started a spinpolarized calculation but the result is not spin polarized.

A: The symmetry between both spin directions has to be broken by hand. Set the `initial polarization` to `'t'`. You can set the amount of the initial split in the alternative submenu `Initial spinsplit`. **Attention:** If there are already two spin sorts in the density file, the program will ask if the splitting shall be skipped (which is correct in most cases). However, in the case discussed in paragraph 4.4.2 a resplit was intended. If the job runs in background, it will abort as soon as the question has to be answered since a background job has normally no standard input to read from.

Q: I have started a spinpolarized calculation with initial spin split and the iteration jumps around.

A: May be the attractor regions for the polarized and for the non polarized solutions are selected in consecutive iteration steps and so it jumps. Depending on the given situation

1. one may try to increase the initial spin split.
2. the iteration-mixing may be too large. Got to alt-submenu `iteration` and decrease it.
3. use FSM to preconverge near the expected moment and release the FSM condition after convergence of the FSM calculation.
4. use FSM to scan to whole $E(M)$ curve (expensive, but sometimes the only way).

B Summary of Changes

01.07.2005:

1. file `=.ldos` is obsolete now, added to FEDIT menu
2. automatic Ewald parameter and Fourier component choice implemented
3. Orbital moment output for full-relativistic calculations
4. **Experimental:** Core-confinement for $4f$-systems
5. minior changes in atom potential to increase stability.

01.07.2004:

1. Full relativistic mode implemented (not for CPA and LSDA+U).
2. New internal mesh settings for increased accuracy.

3. Accuracy improvements in various places.
4. CPA and LSDA+U work together (not full relativistic).
5. Basis orbital grouping implemeted for enhanced stability and avoiding of fixed x_0.
6. Large-scale code restructuring in symmetry treatment in order to implement the full relativistic mode.
7. Format of `=.dens` changed.

03.03.2003:

1. New shape function implemented, which should make voronoi cell merging obsolet. This has an influence on the total energy within the mHartree range.
2. New spherically averaged crystal potential (orbital calculation) introduced, which now is the default. This changes the basis definition and therefore the values of the optimum x_0. It results in a change of total energies. For 3d metals the new basis is a bit worse than the old. However, inclusion of 4d polarization states will, as before, converge the basis to a high degree. For more open structures like oxides the new basis scheme is more stable and in many cases better.
3. CPA implemented.
4. LSDA+U implemented.

31. Oct. 2002:

1. A single step calculation (niter=1) will not overwrite the `=.dens` and `=.basis` files
2. A bug in the symmetrytest is fixed.
3. Enhanced accuracy in the twocenter integrals.

10. Jun 2002:

1. Change in scalar relativistic definition. (Tiny total energy differences < 1 mHa expected.)
2. Kinetic energy correction due to finit radius of orbitals added. Changed results expected, if the basis contains orbitals with large non-vanishing derivative at the outermost mesh point.

06. May 2002:

1. Bugfix for COMPAQ-fort Linux-alpha compiler in bzone.f90.

12. Mar 2002:

1. Definition of empty sites changed: The addition of empty sites to a normal structure will no more change the basis definition of the normal atoms. Thus, the addition of empty sites has to lower the total energy (which was not the case before)!
2. Total crystal density may be output using a grid definition file `=.densgrid`. See Section 3.